Secure static content delivery for content distribution network using blockchain technology

Pier Paolo Tricomi
Student ID: 1179740

***Abstract*—A Content Distribution Network (CDN) is a new kind of network with the goal to distribute services and contents spatially relative to end-users to provide high availability and high performance. Several replicas of the Origin Server are used to reach this goal, but trust issues are now involved both between servers and among client and server. In this work is presented a new method to provide secure static content delivery using blockchain, a growing technology with the capability to ensure reliability and trust without a central authority.**
Moreover, a prototype of the sistem has been developed on Ethereum private network, in order to test the feasibility. The test shows the goodness of the system, and the possibility to create a new content distribution model on the Internet.

1. Introduction

Contents distribution is one of the most important aspects of the Internet. To improve the distribution the Content Distribution Networks (CDNs) are born. CDNs provide high availability and high performance because many replicas (Edge Servers) of the Origin Server are spatially distributed to faster fulfill a request. Many services like CoralCDN [1] CoDeeN [2] provide the possibility to create a CDN starting from the Origin Server, replicating contents and distributing them to end users from the best Edge Server, likely the geographically nearest. A typical scenario can be seen in Figure 1, the Origin Server has more replicas to distribute contents.

In this scenario, two new trust issues are involved. First of all, an attacker could modify the contents from the Origin Server to the Edge Server, thus if the indirectly attacked Edge Server serves that modified content it will be labeled as a misbehaving replica. Secondly, if an Edge Server is not directly managed by the owner of the Origin Server, it can serve different content like stale content or outright modified content, adding ads for example. Furthermore in this architecture the Origin Server is a single point of failure. If the Origin server is compromised all the Servers will misbehave.
The main contribute of this work is two-fold.
First, it is suggested a new architecture to overcome the trust issue between servers, mainteining the CDN properties. Second, the proposed system provides a secure method to deliver static contents, ensuring the integrity with small effort on Clients and Network.

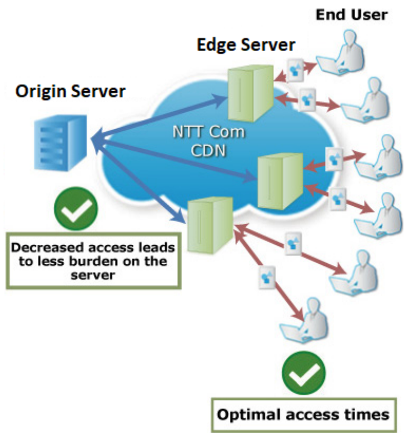

Figure 1. Typical CDN scenario where Origin Server has many Edge Servers to fulfill the clients requests.

2. Related Works

Checking the integrity of data retrieved from untrusted servers is a crucial problem nowadays. The typical approach for preventing contents tampering between clients and servers is to encrypt the end-to-end connection, for example, using the SSL protocol. This negates the functionality of the CDN, because the connection always requires the Origin Server, and the network is no more distributed. For static content, Merkle tree authentication [3] is a possible solution, where the server signs the content which is verified by the client. Also digital rights management schemes allows clients to verify data from untrusted server [4]. For dynamic content, [5] [6] propose the use of XML-based rules for managing the content, but it has to be limited to be easily verified by a client.
In peer-to-peer CDNs the problem is even more significant. LOCKSS [7] uses voting system for content integrity. Repeate the execution to detect misbehavior has been used in Rx [8] and in Vigilante [9] to discover bugs and worms respectively. These approachs require significant overhead on the client. In Pioneer [10] the verify effort is on the dispatcher, a trusted platform, but since the proof of correctness is extremely time sensitive it is not suitable for large scale systems. Finally Repeate and Compare system [11] is for both static and dynamic contents, it requires the repetition of the content to another replica and compares the results

to detect misbehaving replicas. However, the process has significant overhead, and the verify requests may overwhelm the network, thus only a fraction of the contents are verified. In this work blockchain technology is used. The structure of blockchain ensure trust between untrusted nodes without central authority. A system of contents distribution built over blockchain inherits all its benefits, making simple to share contents and verify them between clients and servers.

3. Blockchain Technology

After the rise of Bitcoin [12], its underlying structure, the blockchain technology, has been applied to a variety of usecases ranging from authentication [13] (using Ethereum and Smart Contracts) to medical reports [14]. Blockchain is based on append-only ledger, a growing list of records (called blocks) which are linked and secured using cryptography. The ledger can be viewed by all participating nodes and the updates are permitted only after the consensum of the network. Each block typically contains a cryptographic hash of the previous block, a timestamp and transaction data. Thanks to its structure and the proof-of-work required to create a new block, a blockchain is inherently resistant to modification of the data. A simple example of blockchain is shown in Figure 2.

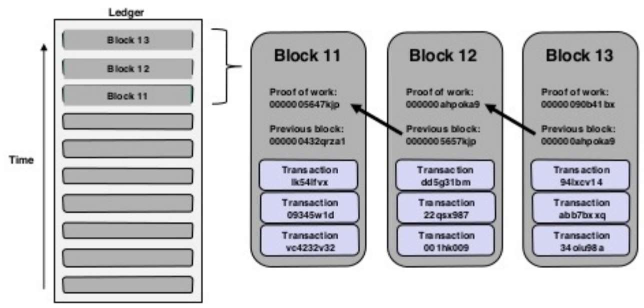

Figure 2. Example of blockchain. Transactions are stored in the blocks of the Ledger. Blocks are linked by hashes.

The presented system relies on Ethereum, an open-source, public, blockchain-based distributed computing platform, which allows to execute smart contracts, immutable programs always visible from the community.

3.1. Ethereum

Ethereum [15] is described as a transactional singleton machine with shared-state. The distributed ledger can be view as a distributed virtual machine which records transactions and state changes. The blockchain is kept alive by the miners, nodes which have three main tasks:

- Verifing the transactions being sent, through the consensum algorithm which is similar to other blockchains, removing the need of central authority;
- Checking if the sender has sufficient gas, the Ethereum's native cryptocurrency, to transfer to the receiver;
- Running the function called by the sender and thus to modify the blockchain state accordingly.

In regards to the third point, we see the most important feature of Ethereum: encoding smart contracts.

3.1.1. Smart Contracts. Smart contracts describe functions, which can be called by nodes participating in the blockchain, and states which can be changed by the nodes. The functions run in the Ethereum Virtual Machine (EVM) which is described as a *quasi*-Turing complete computer able to execute code of any complexity. The *quasi* qualification comes from the fact that the computation is intrinsically bounded to *gas*, which limits the total amount of computation done. If the gas transferred to the miner for the function execution is less than the required amount, then the transaction is not mined.
The contract can use memory and storage to save data. It can use any amount of memory (paying gas) during executing its code, but when execution stops, the entire content of the memory is wiped. The storage on the other hand is persisted into the blockchain itself. It can be changed, but all the changes will be recorded in the ledger.
In the presented sytem, thanks of its persistence and (im)mutability, storage will be used to store and share contents.

3.2. Benefits

In this paragraph, the benefits of using blockchain are discussed. Building the content delivery system on the blockchain it will inherit all the benefits.

- **Decentralization:** Consensus mechanism is used to agree on the validity of transactions, thus there is no need for a trusted third party;
- **Transparency and trust:** Blockchains are shared and visible from all the partecipants. This makes the system transparent and as a result trust is established;
- **Immutability:** Once the data has been written into the blockchain, it is extremely difficult to change it back. In Ethereum if the state is changed all the updates are stored in the ledger. If only few user are allowed to change the state, the state remains immutable as long as they don't change it;
- **High availability:** As the system is based on several nodes in a peer-to-peer network, and the data is replicated and updated on each node, the network as a whole continues to work even if nodes leave or become unavailable;
- **Highly secure:** All transactions on a blockchain are cryptographically secured and provide integrity.

4. System Design

In this section is presented the developed system to distribute and check integrity of the contents. Firstly an high-level overview is shown. Secondly more details about the architecture are provided. Lastly benefits and drowbacks of the system are discussed.

4.1. Overview

Figure 3 shows high-level view of how the presented system distributes the contents on the Servers and permits to Client to check integrity.

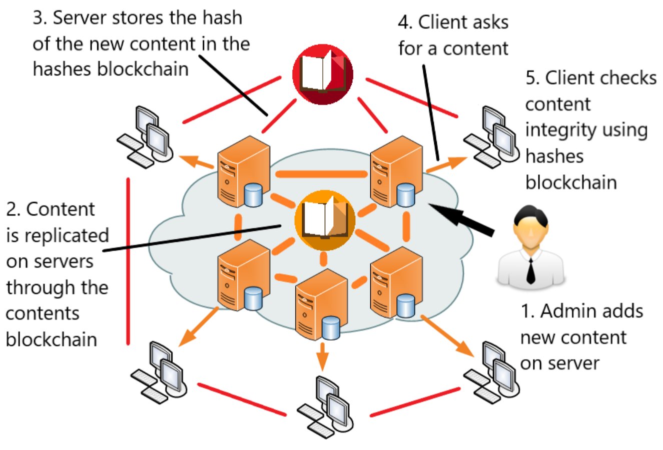
3. Server stores the hash of the new content in the hashes blockchain
4. Client asks for a content
5. Client checks content integrity using hashes blockchain
2. Content is replicated on servers through the contents blockchain
1. Admin adds new content on server

Figure 3. Overview of the content delivery and check system.

The system wants to resolve trust issues between Origin and Edge Servers, since the last can receive altered contents, and between Servers and Clients, as Server can send different content from the requested one. The key idea is to use two different blockchains: the *Contents Blockchain* and the *Hashes Blockchain*.

4.1.1. Contents Blockchain. The proposal is to build the CDN on a blockchain, the *Contents Blockchain*. This means each Edge Server is a node of the blockchain wherein contents will be stored. The Origin Server is no more required, since it is possible to add a content from every node (Edge Server) as long as it is authorizated, and the content will be automatically distributed on each Server thanks to blockchain technology. To authorize the upload of the contents an user system is used. Since the blockchain is always visible from every partecipant, and every partecipant has a public address, which is a public and private key pair, it is enough to store in the blockchain the addresses of who can upload the contents, and after checking who is requesting the upload, the transaction can be executed or not. Two kind of user are permitted: Admin and Superadmin. Admins can upload/remove/edit the contents while Superadmin can also create other Admins.

4.1.2. Hashes Blockchain. The *Hashes Blockchain* is the key to check the integrity of received contents. Thanks to hash functions is really simple to check message integrity [16]. An hash function is any function that can be used to map data of arbitrary size to data of fixed size. A cryptographic hash function allows one to easily verify that some input data maps to a given hash value, but if the input data is unknown, it is extremely difficult to reconstruct it by knowing the stored hash value. These property are very useful for the goal. Every time a content is uploaded, its hash is calculated by the Server and stored in the *Hashes Blockchain*, where the partecipants are both Servers and Clients. After the client receives the requested content it can easily check its integrity calculating its hash and comparing it with the one stored in the blockchain. If the hashes are equals the content is right, otherwise something wrong happend. Since storing the hash of a content requires the consensum of the network, it is always correct, and there is no possibility for a server to change it without noticing the others nodes.

4.2. Typical Usage

A typical usecase is shown in Figure 3. First an Admin wants to upload a new content. After the system checks and authorizes him, a new upload request is sent among the *Contents Blockchain*. When the network approves the request, a copy of the content is distributed on each server (according to blockchain technology) and its hash is calculated and stored in the *Hashes Blockchain*. At any time a client can ask for that content. When the content is received the client calculate its hash with the same hash function used by the Server and compares the obtained hash with the calculated one. If they are equals the integrity is verified, otherwise the client can reject the content and ask to another server.

4.3. Benefits and Drawbacks

In many distributed systems multiple entities maintain their own databases and data sharing can become very difficult due to the disparate nature of the systems. Since a blockchain can serve as a single shared ledger among interested parties, this can reduce the complexity of managing the whole system, removing the possibile processes of verification, reconciliation, and clearance. The contents on all the Servers are always correct and updated, the problem of a replica receiving wrong contents from the main server is avoided.
Many of cited systems like Pioneer [10] and Repeat and Compare [11] require sending multiple request to the server (to get the content and to check it) with the possibility to overcrowd the network, whereas others system requires significant overhead on the client. In the presented system the needed traffic to check integrity is correlated to the mainteining of the blockchain itself, which is not really relevant, and the effort on the client is minimum as shown in Section 5. When client wants to verify the obtained data it just has to calculate an hash, and check the result. Since the

checking process is unexpensive for clients and not stressful for the network, each content can be verified, unlike the other systems where the detection is probabilistic. Furthermore, even the misbehaving Servers detection is simpler thanks to blockchain technology.
Another point of strenght is the removal of the Origin Server, an admin can simply upload a content from his home and it will be distributed among the Servers (peers). Now there is not a single point of failure, since if a Server is compromised the network will see it. The attacker has to steal the private key of an admin to take control of a Server, which is difficult, and it is not a new problem.
In these system we have some drawbacks aswell. First of all, all the Servers must have all the contents that the system can serve, which can be expensive in some scenarios. However, if the *Contents Blockchain* would only contains more requested contents or the URLs of the resources, it is possible to reduces the costs. Secondly, the clients must have a proxy or something similar to automatically check the integrity, but this is not a big deal since it can be done with a simple browser add-on for example. Lastly, keeping updated the blockchain has some costs for the network, but precise tests should be executed to evaluate if this is a real problem, and in a real enviroment the uploading process could be slow due to the mining operation.

5. Implementation

The system relies on Ethereum Blockchain and Smart Contracts. Each Smart Contract can have multiple instance. The contracts are written in Solidity, a high level language designed to encode contracts in Ethereum. The system was tested on a private Ethereum testnet, which is useful for beginners because the use of money (gas) is not involved. To create the network *geth* was used, the the command line interface for running a full ethereum node implemented in Go. The network was composed by few nodes, one Admin and one Superadmin, and a single miner. The difficulty of minining a new block was low, so the testing could be faster. Different Smart Contracts are developed for *Contents Blockchain* and *Hashes Blockchain*. The deploy of contracts was made by Remix IDE, which allows to connect to the private testnet via JSON-RPC.

5.1. Contents Blockchain

The *Contents Blockchain* is where the contents are stored in the Servers. Three Smart Contracts have been developed for the scope:

- **StoredContent**: each instance of this contract represents a content. Every contents has the following attributes (stored in the Storage):
  - the owner, that is the uploader Admin;
  - the name;
  - the description;
  - the timestamp of the upload;
  - the data;
  - the hash.

  Only Admin and Superadmin are allowed to create and manage contents. The contract permits to edit the attributes except the owner, the timestamp and the hash. The last is calculated with *keccak256*, which is an alias of *sha3* supported by Ethereum;
- **AdminAddresses**: only one instance of this contract is required. It has a map containing the addresses of Admins and Superadmins. Methods to check if an address is related to an Admin and which privileges he has are provided. It is also possible for Superadmins to add or remove Admins using this contract;
- **MapContents**: only one instance of this contract is required. It has the map of all the contents, wherein the key is the name of the content. Mapped to the key there are the address of the content, its hash, and the names of the others contents related to it (further information are provided in Section 5.2). Calling this contract an admin can add/delete/update any content.

5.2. Hashes Blockchain

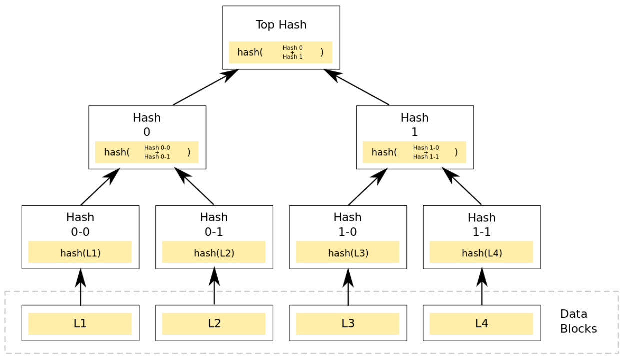

Figure 4. Example of Merkel Tree. In the root is stored the combination of the hashes stored in the leafs.

Assign to clients the task of verifying contents integrity through hashes would mean that the hash of every content has to be in the *Hashes Blockchain*. Actually, Merkel Tree is a perfect data structure for our goals. As shown in Figure 4, a Merkel Tree is a tree in which every leaf node is labelled with the hash of a data block and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. It allows efficient and secure verification of more contents at the same time. Instead of storing one hash per content, using a Merkel Tree of four leafs, for example, is possible to store one hash every four contents. The contents per tree can be raised, and it means a lot of space saving. To check if a content is correct, all the names of the contents of the related Merkel Tree are fetched in the right order. Then it is sufficient to calculate the root hashes starting from the calculated hash of the received content with the hashes of

the other contents in the tree, and in the end comparing the result with the root hash stored in the blockchain.
Two Smart Contracts have been developed for the scope:

- **ClientMerkelTree**: each instance of this contract has the root hash of a Merkel Tree. The contract provides method to calculate, store and check the root hash of four contents.
- **ClientMap**: only one instance of this contract is required. It has the map of all the contents, wherein the key is the name of the content. Mapped to the key there are the address of the related Merkel Tree, the hash of the content and the names of the related contents.

6. Future Works

The presented system has been developed on Ethereum, but it limits the code execution of Smart Contracts imposing gas payment. Some instruction like loops can be very expensive, thus it may not be the best enviroment for the scope. A new ad-hoc blockchain could be implemented for the content distribution, to better fit the requirments. Another improvment could be substituting the data of the content in the *Contents Blockchain* with an URI or another type of storing to permit Servers to have also other contents, depending on the situation. The idea of using a secure URI would lead to create of a global blockchain for contents distribution, where different services and websites can entrust their contents. For example, every service can have its own blockchain, and in the global blockchain could be links to these blockchains. In the end, the presented system is only for static content, a better implementation could be thought for sharing dynamic contents too.

7. Conclusion

In the presented system trust issues in a Content Distribution Network have a solution. Thanks to the growing blockchain technology and its security characteristics, it is possible to secure distribute contents among Servers, and a client can check if the requested content is right. Many problems of exiting systems like strong efforts on the Network or Clients are resolved, and the prototype build on private Ethereum testnet shows the feasibilty of the system. The use of Merkel Tree permits space saving, and other improvement are possible to make this system alive.

Acknowledgments

The authors would like to thank...

References

[1] M. J. Freedman, E. Freudenthal, and D. Mazieres, “Democratizing content publication with coral.” in *NSDI*, vol. 4, 2004, pp. 18–18.

[2] L. Wang, V. Pai, and L. Peterson, “The effectiveness of request redirection on cdn robustness,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 345–360, 2002.

[3] R. J. Bayardo and J. Sorensen, “Merkle tree authentication of http responses,” in *Special interest tracks and posters of the 14th international conference on World Wide Web*. ACM, 2005, pp. 1182–1183.

[4] A. Adelsbach, M. Rohe, and A.-R. Sadeghi, “Towards multilateral secure digital rights distribution infrastructures,” in *Proceedings of the 5th ACM workshop on Digital rights management*. ACM, 2005, pp. 45–54.

[5] C.-H. Chi and Y. Wu, “An xml-based data integrity service model for web intermediaries,” in *Proc. 7th IWCW*, 2002.

[6] H. K. Orman, “Data integrity for mildly active content,” in *Active Middleware Services, 2001. Third Annual International Workshop on*. IEEE, 2001, pp. 73–77.

[7] P. Maniatis, D. S. Rosenthal, M. Roussopoulos, M. Baker, T. J. Giuli, and Y. Muliadi, “Preserving peer replicas by rate-limited sampled voting,” in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 44–59.

[8] F. Qin, J. Tucek, J. Sundaresan, and Y. Zhou, “Rx: treating bugs as allergies—a safe method to survive software failures,” in *Acm sigops operating systems review*, vol. 39, no. 5. ACM, 2005, pp. 235–248.

[9] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, “Vigilante: End-to-end containment of internet worms,” in *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5. ACM, 2005, pp. 133–147.

[10] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, “Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems,” in *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5. ACM, 2005, pp. 1–16.

[11] N. Michalakis, R. Soulé, and R. Grimm, “Ensuring content integrity for untrusted peer-to-peer content distribution networks,” in *Proceedings of the 4th USENIX conference on Networked systems design & implementation*. USENIX Association, 2007, pp. 11–11.

[12] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[13] R. Sundararajan and S. K. Shukla, “Online identity and authentication using blockchain.”

[14] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, “Medrec: Using blockchain for medical data access and permission management,” in *Open and Big Data (OBD), International Conference on*. IEEE, 2016, pp. 25–30.

[15] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[16] G. Tsudik, “Message authentication with one-way hash functions,” in *INFOCOM’92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE*. IEEE, 1992, pp. 2055–2059.